# What is Memory Management?

Memory management is a process of managing computer memory, assigning the memory space to the programs to improve the overall system performance.

## Why is memory management required?

As we know that arrays store the homogeneous data, so most of the time, memory is allocated to the array at the declaration time. Sometimes the situation arises when the exact memory is not determined until runtime. To avoid such a situation, we declare an array with a maximum size, but some memory will be unused. To avoid the wastage of memory, we use the new operator to allocate the memory dynamically at the run time.

## Memory Management Operators

In [C language](https://www.javatpoint.com/c-programming-language-tutorial), we use the **malloc()** or **calloc()** functions to allocate the memory dynamically at run time, and free() function is used to deallocate the dynamically allocated memory. [C++](https://www.javatpoint.com/cpp-tutorial) also supports these functions, but C++ also defines unary operators such as **new** and **delete** to perform the same tasks, i.e., allocating and freeing the memory.

## Why Do We Need Memory Management, and How Does It Work?

## Memory management is required to ensure that there is no wastage of memory and that allocation takes place efficiently. The memory that a C++ program uses is divided into different parts. Here, we will discuss two, i.e. stack and heap.

* Stack: In stack, all the variables that are declared inside the function and other information related to the function are stored.
* Heap: Heap is unused memory and the part from where the memory is allocated dynamically when the program runs.

During the declaration of an [array](https://www.simplilearn.com/tutorials/cpp-tutorial/cpp-array), there are times when the correct memory is not determined until runtime. To avoid such scenarios, we usually declare an array of the maximum size. However, because of this full size, some memory remains unused. For example, let us suppose we have declared an array of size 30, and after declaring the array, it turns out that we only need space of 10 size, so the rest of the space is of no use, or we can say it will get wasted.

To avoid such cases, we use memory allocation. We can allocate the memory at runtime from the heap using an operator.

## Allocation and Deallocation of Memory

Other [programming languages](https://www.simplilearn.com/tutorials/programming-tutorial/first-programming-language) like [java](https://www.simplilearn.com/tutorials/java-tutorial/what-is-java), [python](https://www.simplilearn.com/learn-the-basics-of-python-article), etc., don’t need to allocate memory dynamically. In [C language](https://www.simplilearn.com/c-programming-article), we use the malloc() or calloc() functions to allocate the memory dynamically at run time, and C++ also supports these functions. But, in C++, allocation and deallocation are done manually.

In C++, two operators are used for the allocation and deallocation of memory i.e

* new operator
* delete operator

So, let’s understand the purpose of these two operators.

### C++ New Operator

Using the C++ new operator, we can allocate memory at the runtime. The new operator in C++ is used for the dynamic memory allocation; It is used to allocate the memory at runtime on heap memory.

#### Syntax:

1. pointer\_variable = **new** data-type

The above syntax is used to create the object using the new operator. In the above syntax, **'pointer\_variable'** is the name of the pointer variable, **'new'** is the operator, and **'data-type'** defines the type of the data.

**Example 1:**

1. **int** \*p;
2. p = **new** **int**;

### How to create a single dimensional array

As we know that new operator is used to create memory space for any data-type or even user-defined data type such as an array, structures, unions, etc., so the syntax for creating a one-dimensional array is given below:

1. pointer-variable = **new** data-type[size];

### Examples:

1. **int** \*a1 = **new** **int**[8];

In the above statement, we have created an array of type int having a size equal to 8 where p[0] refers first element, p[1] refers the first element, and so on.

### Delete operator

When memory is no longer required, then it needs to be deallocated so that the memory can be used for another purpose. This can be achieved by using the delete operator, as shown below:

1. **delete** pointer\_variable;

In the above statement, **'delete'** is the operator used to delete the existing object, and **'pointer\_variable'** is the name of the pointer variable.

In the previous case, we have created two pointers 'p' and 'q' by using the new operator, and can be deleted by using the following statements:

1. **delete** p;
2. **delete** q;

### Advantages of the new operator

**The following are the advantages of the new operator over malloc() function:**

* It does not use the sizeof() operator as it automatically computes the size of the data object.
* It automatically returns the correct data type pointer, so it does not need to use the typecasting.
* Like other operators, the new and delete operator can also be overloaded.
* It also allows you to initialize the data object while creating the memory space for the object.

### Advantages of the new operator

**The following are the advantages of the new operator over malloc() function:**

* It does not use the sizeof() operator as it automatically computes the size of the data object.
* It automatically returns the correct data type pointer, so it does not need to use the typecasting.
* Like other operators, the new and delete operator can also be overloaded.
* It also allows you to initialize the data object while creating the memory space for the object.

### Differences between the malloc() and new

* The new operator constructs an object, i.e., it calls the constructor to initialize an object while **malloc()** function does not call the constructor. The new operator invokes the constructor, and the delete operator invokes the destructor to destroy the object. This is the biggest difference between the malloc() and new.
* The new is an operator, while malloc() is a predefined function in the stdlib header file.
* The operator new can be overloaded while the malloc() function cannot be overloaded.
* If the sufficient memory is not available in a heap, then the new operator will throw an exception while the malloc() function returns a NULL pointer.
* In the new operator, we need to specify the number of objects to be allocated while in malloc() function, we need to specify the number of bytes to be allocated.
* In the case of a new operator, we have to use the delete operator to deallocate the memory. But in the case of malloc() function, we have to use the free() function to deallocate the memory.